```
|        return tree2;
|    }
|
|    pair<Node*, Node*> split(Node* tree, int k) {
|        if (tree == nullptr) {
|            return {nullptr, nullptr};
|        }
|        int t = (tree->l != nullptr ? tree->l->size : 0);
|        if (k <= t) {
|            change_parent(tree->l, nullptr);
|            pair<Node*, Node*> trees = split(tree->l, k);
|            change_left(tree, trees.second);
|            return {trees.first, tree};
|        }
|        change_parent(tree->r, nullptr);
|        pair<Node*, Node*> trees = split(tree->r, k-t-1);
|        change_right(tree, trees.first);
|        return {tree, trees.second};
|    }
|
|    Node* add(Node* tree, int k, Node* vertex) {
|        pair<Node*, Node*> trees = split(tree, k);
|        return merge(merge(trees.first, vertex), trees.second);
|    }
|
|    Node* del(Node* tree, int k) {
|        pair<Node*, Node*> trees1 = split(tree, k);
|        pair<Node*, Node*> trees2 = split(trees1.second, 1);
|        return merge(trees1.first, trees2.second);
|    }
|
|    Node* root(Node* tree) {
|        if (tree == nullptr) {
|            return nullptr;
|        }
|        while (tree->parent != nullptr) {
|            tree = tree->parent;
|        }
|        return tree;
|    }
|
|    int find_pos(Node* tree) {
|        if (tree == nullptr) {
|            return 0;
|        }
|        int ans = (tree->l != nullptr ? tree->l->size : 0);
|        while (tree->parent != nullptr) {
|            if (tree->parent->l != tree) {
|                ans += (tree->parent->l != nullptr ? tree->parent->l->size : 0)+1;
|            }
|            tree = tree->parent;
|        }
|        return ans;
|    }
|
|    Node* find_element(Node* tree, int k) {
|        if (tree == nullptr) {
|            return nullptr;
|        }
|        int t = (tree->l != nullptr ? tree->l->size : 0);
|        if (k == t) {
|            return tree;
|        }
|        if (k < t) {
```

```
|            return find_element(tree->l, k);
|        }
|        return find_element(tree->r, k-t-1);
|    }
```

Persistent_DD

```
|    mt19937 randint(179);
|
|    struct Node {
|        int x, size;
|        Node *l, *r;
|
|        Node(int x1): x(x1), size(1), l(nullptr), r(nullptr) {}
|    };
|
|    void update(Node* tree) {
|        if (tree == nullptr) {
|            return;
|        }
|        tree->size = 1;
|        if (tree->l != nullptr) {
|            tree->size += tree->l->size;
|        }
|        if (tree->r != nullptr) {
|            tree->size += tree->r->size;
|        }
|    }
|
|    Node* copy(Node* tree) {
|        if (tree == nullptr) {
|            return nullptr;
|        }
|        Node* now = new Node(tree->x);
|        now->l = tree->l, now->r = tree->r;
|        update(now);
|        return now;
|    }
|
|    Node* merge(Node* tree1, Node* tree2) {
|        if (tree1 == nullptr) {
|            return tree2;
|        }
|        if (tree2 == nullptr) {
|            return tree1;
|        }
|        if (randint() % (tree1->size+tree2->size) < tree1->size) {
|            Node* now = copy(tree1);
|            now->r = merge(tree1->r, tree2);
|            update(now);
|            return now;
|        }
|        Node* now = copy(tree2);
|        now->l = merge(tree1, tree2->l);
|        update(now);
|        return now;
|    }
|
|    pair<Node*, Node*> split(Node* tree, int k) {
|        if (tree == nullptr) {
|            return {nullptr, nullptr};
|        }
|        int left = (tree->l == nullptr ? 0 : tree->l->size);
|        Node* now = copy(tree);
```

```
|        if (k <= left) {
|            pair<Node*, Node*> trees = split(tree->l, k);
|            now->l = nullptr;
|            update(now);
|            trees.second = merge(trees.second, now);
|            return trees;
|        }
|        pair<Node*, Node*> trees = split(tree->r, k-left-1);
|        now->r = nullptr;
|        update(now);
|        trees.first = merge(now, trees.first);
|        return trees;
|    }
|
|    pair<Node*, bool> change(Node* tree, int pos) {
|        auto trees1 = split(tree, pos+1);
|        auto trees2 = split(trees1.first, pos);
|        Node* will = copy(trees2.second);
|        bool flag = (will->x == 1);
|        will->x = 1-will->x;
|        return {merge(merge(trees2.first, will), trees1.second), flag};
|    }
```

Fenwick

```
|    struct Fen {
|        vector<int> fen;
|        int n;
|
|        Fen(int n1) {
|            n = n1+1;
|            fen.assign(n, 0);
|        }
|
|        void plus(int q, int x) {
|            for (++q; q < n; q += (q & -q)) {
|                fen[q] += x;
|            }
|        }
|
|        int sum(int q) {
|            int res = 0;
|            for (; q > 0; q -= (q & -q)) {
|                res += fen[q];
|            }
|            return res;
|        }
|
|        int sum(int l, int r) {
|            return sum(r)-sum(l);
|        }
|    };
```